SQUAD UP

Technical Feasibility Report

Submitted by : Swastik Nagpal

Devansh Wadhwani

Karamjit Malik

Shreya

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

Squad Up is a comprehensive full-stack web application designed to facilitate team formation for various collegiate events including hackathons, technical fests, competitions, and workshops. The platform serves two primary user groups: students seeking to form teams for participation in events, and event organizers managing competitions and participant coordination.

The application leverages modern web technologies including React for frontend development, Node.js with Express for backend services, and MongoDB for data persistence. The system incorporates real-time communication capabilities through Socket.IO, enabling instant messaging and notifications within team environments.

This feasibility study demonstrates that the project can be successfully implemented by a team of one to three students within an 6 to 8-week timeframe, requiring only fundamental knowledge of JavaScript, React, Node.js, and MongoDB. The project architecture has been designed to utilize industry-standard free-tier cloud services, making it economically viable for academic purposes while providing practical experience with production-grade deployment practices.

2. TECHNICAL FEASIBILITY

2.1 Technology Stack

The application will be built using the MERN (MongoDB, Express, React, Node.js) technology stack, chosen for its widespread adoption, extensive community support, and suitability for full-stack JavaScript development.

Component	Technology	Purpose
Frontend Framework	React with Vite	User interface and client-side logic
Styling	TailwindCSS	Responsive UI design
Backend Runtime	Node.js	Server-side JavaScript execution
Web Framework	Express.js	RESTful API development
Database	MongoDB Atlas	Document-oriented data storage
ODM	Mongoose	Schema definition and validation
Authentication	JWT	Stateless authentication mechanism
Real-time Communication	Socket.IO	WebSocket-based messaging

Rationale for Technology Selection:

1. **JavaScript Full-Stack:** Unified language across frontend and backend reduces context switching and learning curve.
2. **Vite:** Provides significantly faster development server startup and hot module replacement compared to traditional bundlers.
3. **TailwindCSS:** Utility-first CSS framework enables rapid UI development without writing custom CSS.
4. **Express.js:** Minimal, unopinionated framework with extensive middleware ecosystem.
5. **MongoDB:** Document model aligns naturally with JavaScript objects, flexible schema suitable for evolving requirements.
6. **Socket.IO:** Abstracts WebSocket complexity while providing fallback mechanisms for browser compatibility.

2.2 System Architecture

The application follows a three-tier architecture pattern separating presentation, application logic, and data management concerns.

Figure 1: System Architecture Overview

Presentation Layer (Client):

React-based single-page application communicating with backend via RESTful APIs. WebSocket connection established for real-time features.

Application Layer (Server):

Express.js server handling HTTP requests through route controllers. JWT middleware for authentication. Socket.IO gateway managing WebSocket connections and event broadcasting.

Data Layer:

MongoDB database accessed through Mongoose ODM. Collections include users, events, teams, messages, and notifications.

Communication Patterns:

- **RESTful APIs:** Used for CRUD operations on resources (users, events, teams, notifications).
- **WebSocket (Socket.IO):** Employed for real-time features including team chat, presence indicators, and instant notifications.
- **JWT Authentication:** Access tokens (15-minute expiry) for API authorization, refresh tokens (7-day expiry) for token renewal.

2.3 Data Model

The database schema comprises five primary collections designed to support the application's core functionality.

Collection	Key Attributes	Relationships
users	_id, name, email, passwordHash, skills, interests, avatar	One-to-many with teams, messages, notifications
events	_id, title, type, rules, teamSizeMin, teamSizeMax, deadline, eligibility, organizerId	References user (organizer), onetomany with teams
teams	_id, name, eventId, members, roles, requirements, status	References event, contains user references
messages	_id, teamId, senderId, text, createdAt	References team and user (sender)
notifications	_id, userId, type, data, read, createdAt	References user (recipient)

Design Considerations:

- Document embedding used for performance-critical queries (team members array).
- References employed for many-to-many relationships to maintain data integrity.
- Timestamps (createdAt, updatedAt) automatically managed by Mongoose.
- Indexes created on frequently queried fields (email, eventId, teamId) for query optimization.

2.4 Security Implementation

Security measures have been incorporated at multiple layers to protect user data and prevent common vulnerabilities.

Authentication and Authorization:

1. **Password Security:** bcrypt hashing with 10 salt rounds applied to all passwords before storage.
2. **JWT Tokens:** Dual-token system with short-lived access tokens and longer-lived refresh tokens stored securely.
3. **Role-Based Access:** Middleware validates user permissions for protected routes.

Input Validation:

Schema validation using Zod or Joi libraries on all incoming requests. Sanitization of user inputs to prevent NoSQL injection attacks. File upload restrictions enforced (type, size) for avatar images.

Rate Limiting and DoS Prevention:

1. Authentication endpoints limited to 5 attempts per 15-minute window per IP address. General API rate limiting applied to prevent resource exhaustion.

Transport Security:

1. HTTPS enforced on all production endpoints through hosting platform configurations. Secure cookie attributes (HttpOnly, Secure, SameSite) for refresh token storage.

Environment Security:

1. Sensitive credentials stored in environment variables, never committed to version control.
2. Different configuration files for development, testing, and production environments.

3. OPERATIONAL FEASIBILITY

Team Structure and Resource Requirements

The project has been scoped to be manageable by a small team of students with foundational web development knowledge. **Team Composition:**

- Team size: 1-3 students
- Required skills: Basic JavaScript, fundamental understanding of React and Node.js
- Time commitment: 5-8 hours per week per team member during active development

Maintenance Requirements

Post-deployment maintenance involves minimal time investment, primarily focused on monitoring and user support.

Maintenance Activity	**Frequency**	**Estimated Time**
Server log review and error monitoring	Daily	10 minutes
User issue response and support	As needed	20 minutes daily average
Content moderation review	Daily	10 minutes
Database backup verification	Weekly	5 minutes

Content Moderation Strategy

Basic moderation capabilities will be implemented to maintain platform integrity:

1. **Automated Filtering:** Integration of profanity filter library for chat messages.
2. **User Reporting:** Report button functionality allowing users to flag inappropriate content.
3. **Administrative Controls:** Simple admin interface enabling organizers to mute or remove users from specific events.

4. **Audit Trail:** Logging of moderation action for accountability and review purposes.

Knowledge Transfer and Documentation

Comprehensive documentation will be maintained throughout development to facilitate knowledge transfer and future maintenance:

Code comments and inline documentation for complex logic.
API documentation detailing all endpoints, request/response formats, and authentication requirements.

Deployment guide with step-by-step instructions for reproducing the production environment.

Troubleshooting guide addressing common issues encountered during development.

4. SCHEDULE FEASIBILITY

The project timeline has been structured as an 8-week development cycle, with each week focusing on specific modules and deliverables. The schedule assumes 6-10 hours of work per week per team member.

Detailed Development Timeline

Week 1: Foundation and Infrastructure Setup

Initialize Git repositories for frontend and backend

Configure development environment and tooling

Set up continuous integration pipelines

Define base Mongoose schemas for User, Event, and Team models

Week 2: Authentication and User Management

Implement user registration with input validation

Develop login system with JWT token generation

Create logout and token refresh mechanisms

Build user profile management (view, update)

Implement skills and interests functionality

Integrate avatar upload (optional Cloudinary integration)

Develop protected route middleware for React

Week 3: Event Management Module

Create event creation interface for organizers

Implement event listing with pagination

Develop search and filter functionality (by type, deadline)

Build detailed event view displaying rules, requirements, and eligibility

Implement event update and deletion capabilities for organizers

Week 4: Team Formation and Management

Develop team creation workflow

Implement join request and approval system

Create leave team functionality

Build team listing page with filters (by event, required skills)

Implement team member invitation system

Week 5: Team Roster and Notification System

Develop team roster display with member roles

Implement role assignment functionality

Create team requirements specification interface

Build in-app notification system (database-backed)

Implement read/unread notification status

Develop notification centre interface

Week 6: Real-time Communication

Integrate Socket.IO on client and server

Implement team-specific chat rooms

Develop message persistence to MongoDB

Create chat history retrieval functionality

Implement typing indicators

Add online/offline status indicators (basic implementation)

Week 7: Advanced Features and Organizer Dashboard

Implement email notification system (optional via Brevo)

Develop organizer dashboard for event management

Create team approval workflow for organizers

Build participant list view

Implement basic analytics (participant counts, team statistics)

Add event status management (active, completed, cancelled)

Week 8: Polish, Testing, and Documentation

Implement responsive design across all screen sizes

Add loading states and skeleton screens

Develop comprehensive error handling and user feedback

Create seed data script for demonstration purposes

Set up demo accounts with predefined data

Complete technical documentation (ERD, API specification)

Write README with setup instructions and environment configuration

Prepare demonstration script and recording

Critical Path Analysis

The following dependencies must be respected in the development sequence:

1. Authentication system must be completed before implementing any protected features.

2. Event management must precede team formation, as teams are associated with events.
3. Team management must be operational before implementing real-time chat functionality.
4. Basic notification system should be in place before adding email notifications.

Risk Buffer

The 8-week timeline includes implicit buffer time in each week's allocation. If development proceeds ahead of schedule, the extra time can be allocated to:

- Enhanced UI/UX refinement
- Additional testing and bug fixing
- Implementation of stretch features (advanced search, recommendation system)
- Performance optimization